# Practical guide

## File 1: add\_century

1. Requires: ‘totalset.csv’, ‘Metadata\_IMPACT\_books.xlsx', 'MetadataHistorical\_newspaper\_OCR\_groundtruth.xlsx', 'Metadata\_DBNL\_OCR\_v1.xlsx'
2. Merges data and metadata
3. Changes year ‘1637’ into century ‘1600s’ for all documents
4. Returns dataframe ‘df\_centuries.csv'

## File 2: Alignment checker

Assesment of performance of alignment checker.

## File 3: Preprocessing and EDA

1. Requires: 'df\_centuries.csv'
2. Randomly shuffle rows for more diverse distribution over train, validation, and test set
3. Reduce documents.
4. IMPORTANT: for the reducement of documents, there are various parameters that determine the new size of the document. In the appendix of the thesis, the process is described. The amount of pages per century used here is: [0.13932059752944556, 2.94638517618469, 7.68581616481775, 2.8344535359438923] for respectively the 1600s, 1700s, 1800s, 1900s. Eventually of the reduced dataset a sample of 5000 documents was used.
5. Preprocessing of OCR-output and GT: put in lowercase and removal of all punctuation except for hyphens (to preserve hyphenated words as one word) and dots (to keep distinction between sentences).
6. Create train, validation and test set:

The train set contains 60% of each century’s documents.

The validation set contains 20% of each century’s documents.

The test set contains 20% of each century’s documents.

This means that each dataset contains the same distribution of documents over the centuries.

1. Matching of sentences between OCR and GT: in order to remove the GT sentences of which there is no OCR-output, and in order to make alignment on a word level possible, sentences are matched based on a match of at least 75%.
2. Only for documents in the test set, which after correction will be compared to the original documents, a WER and CER is calculated with the IMPACT evaluation tool. This is done over the OCR which had been matched on a sentence level with the GT, since this in the one that will be corrected.
3. Calculation of words, sentences, and sentence lengths in each document.
4. Find the longest sentence: this is useful for the padding of BERT inputs.
5. Word alignment: each word in the OCR-output is matched to a word in the GT
6. Word alignment automatic check: if a sentence is not well enough aligned, it will not be used for the detection and correction task. Calculate percentage of well aligned sentences per century.
7. Count amount of missing words (OCR-output words for which there was no GT word)

after word alignment following each other and on average.

1. Create a list of proper pronouns in the OCR-output and one list of those in the GT. NNP’s shorter than five characters are removed, as these are often insertions between the first and last name (‘van’, ‘de’, ‘der’, ‘den’, ‘vande’), which cause the model to see each appearance of those as a NNP.
2. Returns: ‘preprocessed\_df.csv’, ‘ocr\_names.txt’, ‘gt\_names.txt’

## File 4: Preprocessing for models

1. Requires: 'preprocessed\_df.csv', ‘gt\_names.txt’
2. Use preprocessed GT text in df train set. Returns preprocessed train GT as txt for other files.
3. More preprocessing before training: numbers are replaced with %NUMBER%, proper pronouns are replaced with %NNP% in training data.
4. The training text is fed to word2vec and BERT finetuning in the form of sentences.
5. Finetuning requires CUDA/GPU.
6. Returns: ‘word2vec\_finetuned.model’, 'BERT\_finetuned.pt', ‘gtfortraining.txt’

## File 5: Correction and detection validation

1. Requires: ‘word2vec\_finetuned.model’, 'BERT\_finetuned.pt', 'BERT\_finetuned.pt', ‘ocr\_names.txt’
2. Goal: find optimal parameters for amount of detection candidates and correction candidates, for both word2vec and BERT.
3. def detection\_and\_correction\_word2vec and def detection\_and\_correction\_BERTje:
4. Skip over NNPs, tokens with numbers, tokens of length two and smaller (‘f’ (gulden sign), loose letters etc.) and tokens in bad alignments.
5. Creates list of 1000 candidates which will be used for both detection and correction.
6. For detection parameter analysis: create two lists: one for positions of OCR tokens in candidates list when it is an error,one for positions of OCR tokens in candidates list when it is not an error, for both word2vec and BERT. Plot occurrences in form of percentages, in bins of 10, and find point where these meet to determine topn\_detection parameter for both word2vec and BERT.
7. For correction parameter analysis: create list for positions of correct corrections in candidates list. If an OCR token is an actual error, look for the right correction (matching GT token) in the candidates list. Add position to list. Find topn\_correction parameter for both word2vec and BERT by finding 95th percentile (alpha = 0.05).
8. For correction method analysis: for all three methods (sorted, sorted no stopwords, calculated score), calculate an accuracy for both word2vec and BERT. Choose highest accuracy of these three methods, for word2vec and BERT, as correction method.
9. Returns: 'BERT\_positions.txt', 'word2vec\_positions.txt', 'validation\_BERT.csv', 'validation\_word2vec.csv'

## File 6: Create lists for special tokens

1. Requires: (BERT and word2vec vocab will be reconstructed from tokenizer and the train set), ‘combined-160.txt’/’combined-320.txt’, ‘dictionaryBasedDutchDictionary\_1.0.type\_frequency.txt’ (first list of historical expressions), ‘dutchCorpusBasedDictionary\_1.1.tf.txt’ (second list of historical expressions), ‘wordlist.txt’ (modern vocabulary), ‘homoniemen.txt’, (these lists are from IVDNT).
2. Seven lists are created:

* Homonyms
* vocab\_BERT: all vocabulary of the pretrained model and the finetuned model
* vocab\_word2vec: all vocabulary of the pretrained model and the finetuned model
* hist\_expressions: concatenation of the above mentioned two lists
* modern\_vocab: modern dictionary of 400.000 words
* infrequent\_expressions: expressions that only occur once in the train dataset
* dictionary: concatenation of historical and modern words

1. Returns: these seven lists

## File 7, 8 & 9: Detection and correction test w2v / BERT / baseline

1. Requires: "word2vec\_finetuned.model", 'BERT\_finetuned.pt' (depending on model), 'preprocessed\_df.csv', ‘ocr\_names.txt’
2. Each file performs the same process for one of the different three models:
3. Detection task word2vec / BERT:

a.Skip numbers, NNP’s, short words, bad alignments.

b.. Generate candidate list (with model) of the length of the biggest parameter (of topn\_detection and topn\_correction, determined in file 5). Generate cosines/probabilties and LD list matching the candidates. Cut down the candidates list to the length of topn\_detection.

1. If there are only OOV words in the context for word2vec, the token is detected as an error.
2. If the token is in the candidates list, it is predicted to not be an error. Otherwise, it is predicted to be an error.
3. There is an analysis which keeps track of false positives and negatives and true positives and negatives. This happens for all the tokens, but also specifically for those which are homonyms, OOV, historical expressions, infrequent expressions, and real word errors. This is also done separately when one of those is in the context of the word. Furthermore, when none of those are in the context.
4. Calculate the precision, recall, F1 for all tokens and special tokens.
5. Correction task word2vec/BERT:
   1. Skip numbers, NNP’s, short words, bad alignments.
   2. Correction for the evaluation is done on tokens of which we know are incorrection (different from the ground truth).
   3. Cut down the candidates list, cosines/probabilties to the length of topn\_correction.
   4. If there are only OOV words in the context for word2vec, the token is not corrected.
   5. The token is corrected with the method chosen to be the best in file 5.
   6. There is an analysis which keeps track of the wrong and right corrections. This happens for all the tokens, but also specifically for those which are homonyms, OOV, historical expressions, infrequent expressions, and real word errors. This is also done separately when one of those is in the context of the word. Furthermore, when none of those are in the context.
6. Detection task baseline:
   1. Skip numbers, NNP’s, short words, bad alignments.
   2. If a token is not in the dictionary created in file 5: it is predicted to be an error.
   3. There is an analysis which keeps track of false positives and negatives and true positives and negatives. This happens for all the tokens, but also specifically for those which are homonyms, OOV, historical expressions, infrequent expressions, and real word errors. This is not done for the context, as the lexical method does not consider context.
   4. Calculate the precision, recall, F1 for all tokens and special tokens.
7. Correction task baseline:
   1. Skip numbers, NNP’s, short words, bad alignments.
   2. Correction for the evaluation is done on tokens of which we know are incorrection (different from the ground truth).
   3. An erroneous token is corrected to the word with the closest LD similarity in the dictionary.
   4. There is an analysis which keeps track of the wrong and right corrections. This happens for all the tokens, but also specifically for those which are homonyms, OOV, historical expressions, infrequent expressions, and real word errors. This is again not done for context.
8. Whole task: the model is used as it would be on a new dataset: first detect errors, then correct those which are predicted to be errors.

The evaluation on this task is done by calculating a new WER and CER and comparing to the original WER and CER.

1. Returns: three dataframes: one for detection evaluation, one for correction evaluation and one for whole task evaluation, for each model.